Supplementary Material

We first provide additional results in Appendix A. Then, in Appendix B we provide analyses of the covariance neighborhood size and the effect of training the image encoder. The equations for the different kernels used in our ablation study are shown in Appendix C. In Appendix D, we list the runtimes of the different components in our approach. Appendix E contains pseudo-code for our dense GP, a list detailing the trainable layers of our approach, and the details on how the output of the GP is presented to the decoder. Finally, in Appendix F, we provide a qualitative comparison and qualitative results.

A Additional Results

We provide the per-fold results on the COCO-20^i to PASCAL transfer experiment and a list of the classes included; the per-shot results used to generate Fig. 1 in the main paper; and the per-fold results used in the state-of-the-art comparison.

A.1 Cross-dataset Evaluation

COCO-20^i to PASCAL Transfer We supply additional details on the cross-dataset evaluation experiment in Table 2 of the main paper. This experiment was proposed by Boudiaf *et al.* [4] and we follow their setup. First, our approach is trained on each of the four folds of COCO-20^i. Next, we test each of the four versions on PASCAL, using only the classes held-out during training. We list the classes of each fold in Table 7. The full per-fold results are presented in Table 6.

Table 6. The results of our approach in a COCO-20^i to PASCAL transfer experiment (mIoU, higher is better). Following Boudiaf *et al.* [4], the approach is trained on a fold of COCO-20^i training set and tested on the PASCAL validation set. The testing folds are constructed to include classes not present in the training set, and thus not the same as PASCAL-5^i.

Method	1-Shot					5-Shot				
	F-0	F-1	F-2	F-3	Mean	F-0	F-1	F-2	F-3	Mean
RPMM [40]	36.3	55.0	52.5	54.6	49.6	40.2	58.0	55.2	61.8	53.8
PFENet [31]	43.2	65.1	66.5	69.7	61.1	45.1	66.8	68.5	73.1	63.4
RePRI [4]	52.8	64.0	64.1	71.5	63.1	57.7	66.1	67.6	73.1	66.2
Ours, ResNet50	**55.1 ± 0.8**	**71.0 ± 0.4**	**69.2 ± 0.9**	**80.3 ± 0.8**	**68.9 ± 0.4**	**70.3 ± 0.9**	**75.3 ± 0.4**	**78.5 ± 0.7**	**85.8 ± 0.4**	**77.5 ± 0.2**
Ours, ResNet101	**55.1 ± 0.4**	**72.2 ± 0.3**	**70.7 ± 0.8**	**82.3 ± 0.8**	**70.1 ± 0.3**	**70.7 ± 0.7**	**75.6 ± 0.6**	**80.2 ± 0.1**	**87.3 ± 0.3**	**78.5 ± 0.3**

A.2 1-10 Shot Results

Here, we provide the full results from 1-10 shots, which was used to generate Figure 1. In Table 8 and 9 our results on PASCAL-5^i and COCO-20^i are presented. Note that our approach was trained for 1 shot in the 1-shot setting, and 5 shots in the other nine settings.

Table 7. The classes used for testing in the COCO-20^i to PASCAL transfer experiment, as proposed by Boudiaf *et al.* [4]. This split is different from that of PASCAL-5^i in order to avoid overlap between the training and testing classes.

Fold-0	**Fold-1**	**Fold-2**	**Fold-3**
Airplane,	Boat, Bicycle, Bus, Horse,	Bird, Car, Potted Bottle, Cat, Cow,	
Chair,	Dining Sofa	Plant, Sheep, Train, Motorcycle	
Table, Dog, Person		TV-monitor	

Table 8. 1 to 10 -shot PASCAL-5^i results.

	1-shot	2-shot	3-shot	4-shot	5-shot	6-shot	7-shot	8-shot	9-shot	10-shot
DGPNet (ResNet101)	64.8 ± 0.5	68.4 ± 0.5	72.4 ± 0.3	74.2 ± 0.5	75.4 ± 0.4	76.1 ± 0.4	76.6 ± 0.4	77.0 ± 0.4	77.5 ± 0.4	77.7 ± 0.4

Table 9. 1 to 10 -shot COCO-20^i results.

	1-shot	2-shot	3-shot	4-shot	5-shot	6-shot	7-shot	8-shot	9-shot	10-shot
DGPNet (ResNet101)	46.8 ± 0.3	51.7 ± 0.2	55.0 ± 0.2	56.7 ± 0.3	57.9 ± 0.3	58.7 ± 0.2	59.2 ± 0.3	59.7 ± 0.3	60.0 ± 0.3	60.2 ± 0.2

A.3 Per-Fold Results

In this section, we provide the full results for our state-of-the-art comparison. In Tables 10 and 11 the per-fold results for both ResNet50 and ResNet101 are presented. In general our results are stable for all folds.

Table 10. Per-fold results on PASCAL-5^i

Method	1-Shot 5^0	5^1	5^2	5^3	Mean	5-Shot 5^0	5^1	5^2	5^3	Mean
CANet (ResNet50)	52.5	65.9	51.3	51.9	55.4	55.5	67.8	51.9	53.2	57.1
DENet (ResNet50)	55.7	69.7	63.6	51.3	60.1	54.7	71.0	64.5	51.6	60.5
PFENet (ResNet50)	61.7	69.5	55.4	56.3	60.8	63.1	70.7	55.8	57.9	61.9
RePri (ResNet50)	60.2	67.0	61.7	47.5	59.1	64.5	70.8	71.7	60.3	66.8
ASGNet (ResNet101)	59.8	67.4	55.6	54.4	59.3	64.6	71.3	64.2	57.3	64.4
SCL (ResNet50)	63.0	70.0	56.5	57.7	61.8	64.5	70.9	57.3	58.7	62.9
SAGNN (ResNet50)	64.7	69.6	57.0	57.2	62.1	64.9	70.0	57.0	59.3	62.8
DGPNet (ResNet50)	63.5 ± 0.9	71.1 ± 0.5	58.2 ± 1.6	61.2 ± 0.7	63.5 ± 0.4	72.4 ± 0.8	76.9 ± 0.2	73.2 ± 0.9	71.7 ± 0.4	73.5 ± 0.3
DGPNet (ResNet101)	63.9 ± 1.2	71.0 ± 0.5	63.0 ± 0.6	61.4 ± 0.6	64.8 ± 0.5	74.1 ± 0.6	77.4 ± 0.6	76.7 ± 0.9	73.4 ± 0.6	75.4 ± 0.4

B Detailed Analysis

We present results from two additional experiments. First, the size of the local covariance region is analyzed. Then, we investigate the impact of freezing the backbone during episodic training.

B.1 Additional Covariance Neighborhood Experiments

In section 3.5 we show how the covariance with neighbors in a local region is fed to the decoder. In Table 12, we supply additional results with different sized windows.

Table 11. Per-fold results on COCO-20^i

Method	1-Shot 20^0	20^1	20^2	20^3	Mean	5-Shot 20^0	20^1	20^2	20^3	Mean
DENet (ResNet50)	42.9	45.8	42.2	40.2	42.8	45.4	44.9	41.6	40.3	43.0
PFENet (ResNet50)	36.8	41.8	38.7	36.7	38.5	40.4	46.8	43.2	40.5	42.7
RePri (ResNet50)	31.2	38.1	33.3	33.0	34.0	38.5	46.2	40.0	43.6	42.1
ASGNet (ResNet50)	-	-	-	-	34.6	-	-	-	-	42.5
SCL (ResNet101)	36.4	38.6	37.5	35.4	37.0	38.9	40.5	41.5	38.7	39.9
SAGNN (ResNet101)	36.1	41.0	38.2	33.5	37.2	40.9	48.3	42.6	38.9	42.7
DGPNet (ResNet50)	43.6 ± 0.5	47.8 ± 0.8	44.5 ± 0.8	44.2 ± 0.6	45.0 ± 0.4	54.7 ± 0.7	59.1 ± 0.5	56.8 ± 0.6	54.4 ± 0.6	56.2 ± 0.4
DGPNet (ResNet101)	45.1 ± 0.5	49.5 ± 0.8	46.6 ± 0.8	45.6 ± 0.3	46.7 ± 0.3	56.8 ± 0.8	60.4 ± 0.9	58.4 ± 0.4	55.9 ± 0.4	57.9 ± 0.3

We experiment with $N \in \{1, 3, 5, 7\}$. Larger windows improve the results but the improvement seems to saturate after $N = 5$.

Table 12. Performance for different configurations of covariance windows on the PASCAL-5^i and COCO-20^i benchmarks. Measured in mIoU (higher is better).

				PASCAL-5^i		COCO-20^i		
1x1	3x3	5x5	7x7	1-shot	5-shot	1-shot	5-shot	Δ
				62.1	69.9	41.7	51.2	0.0
✓				60.0	70.3	42.2	51.7	-0.2
	✓			62.0	71.1	43.6	53.0	1.2
		✓		62.5	**71.8**	**43.8**	**53.7**	**1.7**
			✓	**63.0**	**71.7**	**43.7**	**53.5**	**1.8**

B.2 Effect of Training the Image Encoder

In Table 13 we compare our final approach trained with a frozen and unfrozen backbone. Several prior works found it beneficial to freeze the image encoder during episodic training. In contrast, we find it beneficial to not freeze it and keep learning visual representations. By differentiating through our GP learner during episodic training, the proposed method can thus refine the underlying feature representations, which is another important advantage of our approach. However, our approach still improves upon state-of-the-art when employing a frozen backbone.

B.3 Additional Baseline Experiments

We supply three additional baseline experiments to validate the effect of the proposed DGP-module. We evaluate three variants: (i) We remove the GP (the f-branch) and let the decoder rely only on the shallow features. (ii) We replace the GP with the Prior-mask learning mechanism proposed in PFENet. (iii) We replace the GP with a prototype-based approach where the support features are mask-pooled and the result compared to the query features using cosine-similarity, similar to e.g. PANet. Results on PASCAL and COCO are reported in the Table 14. Our approach outperforms all baselines by a large margin. This further validates the superiority of our GP module.

Table 13. Comparison between freezing the backbone and fine-tuning it with a low learning rate.

Backbone	Method	PASCAL-5^i 1-shot	PASCAL-5^i 5-shot	COCO-20^i 1-shot	COCO-20^i 5-shot
ResNet50	DGPNet (Frozen Backbone)	61.9 ± 0.3	72.4 ± 0.3	43.1 ± 0.3	54.5 ± 0.2
	DGPNet (Ours)	**63.5 ± 0.4**	**73.5 ± 0.3**	**45.0 ± 0.4**	**56.2 ± 0.4**
ResNet101	DGPNet (Frozen Backbone)	63.4 ± 0.5	74.3 ± 0.3	44.6 ± 0.5	56.6 ± 0.3
	DGPNet (Ours)	**64.8 ± 0.5**	**75.4 ± 0.4**	**46.7 ± 0.3**	**57.9 ± 0.3**

Table 14. Additional baseline experiments to validate the effect of the DGP-module.

Method	PASCAL-5^i 1-shot	PASCAL-5^i 5-shot	COCO-20^i 1-shot	COCO-20^i 5-shot
No GP	42.9	43.0	23.5	23.5
PFENet prior mask	54.0	54.7	35.8	37.2
Prototype	57.5	63.0	41.5	49.9
DGPNet (Ours)	**63.5**	**73.5**	**45.0**	**56.2**

C Kernel Details

In this section we provide the definitions of the kernels used in our ablation study. First we define the homogenous linear kernel as,

$$\kappa_{\text{lin}}(x, y) = x^T y \ .$$

As was noted in the paper, this kernel corresponds to Bayesian Linear Regression with a specific prior. One could also consider learning a bias parameter, however we chose not to learn such parameters for reasons of method simplicity. Next we consider the exponential kernel,

$$\kappa_{\text{exp}}(x, y) = \exp\left(-\frac{||x - y||_2}{\ell}\right).$$

This kernel behaves similarly as the SE kernel, however with a sharper peak and slower rate of decay. For completeness we additionally define the SE kernel here again,

$$\kappa_{\text{SE}}(x, y) = \exp\left(-\frac{||x - y||_2^2}{2\ell^2}\right).$$

We chose the length of the exponential kernel as $\ell = \sqrt{D}$ and the squared exponential kernel as $\ell^2 = \sqrt{D}$, we found the performance in general to be robust to different values of ℓ. Note that we used the same value of ℓ for both benchmarks and for all number of shots.

In Table 15 we provide a kernel hyperparameter sensitivity analysis. We run one experiment per value and choose the values to cover one order of magnitude centered around the values adopted. Perturbing the hyperparameters with a factor of 0.3 or 3.0 leads to minor but statistically significant drops in performance. One exception is increasing σ_f^2 by a factor of 3.0, which does not adversely affect performance.

Table 15. Kernel hyperparameter sensitivity analysis for the SE-kernel, which is adopted in the main paper. The sensitivity analysis is based on the full, final approach with a ResNet50 backbone and performed on the 5-shot setting in PASCAL-5^i. The hyperparameters values are chosen to cover one order of magnitude of hyperparameter values, centered at the values adopted in the main paper.

$\ell^2 = 0.3\sqrt{D}$	$\ell^2 = \sqrt{D}$	$\ell^2 = 3.0\sqrt{D}$	$\sigma_f^2 = 0.3$	$\sigma_f^2 = 1.0$	$\sigma_f^2 = 3.0$	$\sigma_y^2 = 0.03$	$\sigma_y^2 = 0.1$	$\sigma_y^2 = 0.3$
72.9	**73.5± 0.3**	72.8	72.8	**73.5± 0.3**	**73.6**	73.0	**73.5± 0.3**	72.9

Table 16. Runtimes of the different functions in our approach, measured in milliseconds (ms). Timings are measured in evaluation mode on 512×512 sized images from COCO-20^i.

	Batch size 1		Batch size 20	
	1-shot	5-shot	1-shot	5-shot
Image encoder on support	15.5	23.4	48.5	197.5
Mask encoder on support	2.1	2.1	1.6	6.6
GP preparation on support	8.3	13.9	7.2	113.5
Image encoder on query	9.7	13.6	39.8	39.9
GP inference on query	7.9	9.4	12.1	38.1
Decoder	6.2	6.9	40.0	40.5
Total	49.7 ms	69.3 ms	149.2 ms	436.1 ms

D Runtimes

We show the runtime of our method in Table 16. We partition the timing into different parts. The Gaussian process (GP) is split into two. One part preparing and decomposing the support set matrix in (6) and (7) of the main paper, and another part computing the mean and covariance of the query given the pre-computed matrix decomposition.

The timings are measured in the 1-shot and 5-shot settings on images from COCO-20^i of 512×512 resolution, using either a single episode per forward or a batch of 20 episodes in parallel. We run the method in evaluation mode via `torch.no_grad()`. Timing is measured by injecting cuda events around function calls and measuring the elapsed time between them. The events are inserted via `torch.cuda.Event(enable_timing=True)`. We run our approach on a single NVIDIA V100 for 1000 episodes and report the average timings of each part.

E Additional Implementation Details

We provide code for the Gaussian Process inference, the neural network layers that make up the modules used in our approach, and the details of how the predictive output distribution from the GP is fed to the decoder.

E.1 Code for Gaussian Process

Pseudo-code for the dense GP is shown in Listing 1.1. Equation 7 involves the multiplication with a matrix inverse. It is in practice computed via the Cholesky decomposition and solving the resulting systems of linear equations. For brevity and clarity, we omit device casting and simplify the solve implementation. In practice, we use the standard triangular solver in PyTorch, `torch.triangular_solve`.

```
def GP(x_q, y_s, x_s, sigma_y, kernel):
    """ Produces the predictive posterior distribution of the GP.
    After each line, we comment the shape of the output, with sizes
    defined as:
    B is the batch-size
    Q is the number of query feature vectors in the query image
    S is the number of support feature vectors across the support images
    M is the number of channels in the GP output space
    D is the number of channels in the feature vectors.

    Args:
    x_q: deep query features (B,Q,D)
    y_s: support outputs (B,S,M)
    x_s: deep support features (B,S,D)
    sigma_y: mask standard deviation
    kernel: the kernel function,
    """
    B, S, D = x_s.shape
    I = torch.eye(S)
    K_ss = kernel(x_s, x_s)  #(B,S,S)
    K_qq = kernel(x_q, x_q)  #(B,Q,Q)
    K_sq = kernel(x_s, x_q)  #(B,Q,S)
    L_ss = torch.cholesky(K_ss + sigma_y**2 * I)  #(B,S,S)
    mu_q = (K_sq.T @ solve(L_ss.T, solve(L_ss, y_s))  #(B,Q,M)
    v = solve(L_ss, K_sq)  #(B,S,Q)
    cov_q = K_qq - v.T @ v  #(B,Q,Q)
    return mu_q, cov_q
```

Listing 1.1. PyTorch implementation of the Gaussian Process utilized in the proposed approach. Here, the learning and inference is combined in a single step. The `@` operator denotes matrix multiplication and `solve` the solving of a linear system of equations. The `.T` is the batched matrix transpose.

E.2 Trainable Layer List

In Table 17 we report the trainable neural network layers used in our approach. The image encoder listed is either a ResNet50 [9] or a ResNet101 [9] with two projection

layers. The results of *layer3* and *layer4* are each fed through a linear projection layer, *layer3 out* and *layer4 out* respectively, to produce feature maps at stride 16 and stride 32. These two feature maps are then fed into one GP each in the GP pyramid. The support masks is fed through another ResNet [9] in order to produce the support outputs. The GPs are integrated as neural network layers, but do not contain any learnable parameters. The decoder is a DFN [43]. We do not adopt the border network used in their work and we skip the global average pooling. We also feed it shallow feature maps extracted from the query. The shallow feature maps are the results of *layer1* and *layer2* in the image encoder, at stride 4 and 8 respectively.

Table 17. All neural network blocks used by our approach. The rightmost column shows the dimensions of the output of each block, assuming a 512×512 input resolution. The image encoder is from He *et al*. [9] and the decoder from Yu *et al*. [43]. The `BottleNeck` and `BasicBlock` blocks are from He *et al*. [9], and the `CAB` and `RRB` blocks from Yu *et al*. [43]. See their works for additional details.

Image Encoder		
conv1	`Conv2d`	$64 \times 256 \times 256$
bn1	`BatchNorm2d`	$64 \times 256 \times 256$
relu1	`ReLU`	$64 \times 256 \times 256$
maxpool	`MaxPool2d`	$64 \times 128 \times 128$
layer1	`3x BottleNeck`	$256 \times 128 \times 128$
layer2	`4x BottleNeck`	$512 \times 64 \times 64$
layer3	`6x/23x BottleNeck`	$1024 \times 32 \times 32$
layer4	`3x BottleNeck`	$2048 \times 16 \times 16$
layer3 out	`Conv2d`	$512 \times 32 \times 32$
layer4 out	`Conv2d`	$512 \times 32 \times 32$
Mask Encoder		
conv1	`Conv2d`	$16 \times 256 \times 256$
bn1	`BatchNorm2d`	$16 \times 256 \times 256$
relu1	`ReLU`	$16 \times 256 \times 256$
maxpool	`MaxPool2d`	$16 \times 128 \times 128$
layer1	`BasicBlock`	$32 \times 64 \times 64$
layer2	`BasicBlock`	$64 \times 32 \times 32$
layer3	`BasicBlock`	$64 \times 16 \times 16$
layer2 out	`Conv2d+BatchNorm2d`	$64 \times 32 \times 32$
layer3 out	`Conv2d+BatchNorm2d`	$64 \times 16 \times 16$
Decoder		
rrb in 1	`RRB`	$256 \times 16 \times 16$
cab 1	`CAB`	$256 \times 16 \times 16$
rrb up 1	`RRB`	$256 \times 16 \times 16$
upsample1	`Upsample`	$256 \times 32 \times 32$
rrb in 2	`RRB`	$256 \times 32 \times 32$
cab 2	`CAB`	$256 \times 32 \times 32$
rrb up 2	`RRB`	$256 \times 32 \times 32$
upsample2	`Upsample`	$256 \times 64 \times 64$
rrb in 3	`RRB`	$256 \times 64 \times 64$
cab 3	`CAB`	$256 \times 64 \times 64$
rrb up 3	`RRB`	$256 \times 64 \times 64$
upsample3	`Upsample`	$256 \times 128 \times 128$
rrb in 4	`RRB`	$256 \times 128 \times 128$
cab 4	`CAB`	$256 \times 128 \times 128$
rrb up 4	`RRB`	$256 \times 128 \times 128$
conv out	`Conv2d`	$2 \times 128 \times 128$
upsample4	`Upsample`	$2 \times 512 \times 512$

E.3 Final Mask Prediction Details

In 3.5 we transformed the output of the GP, restoring spatial structure, before feeding it into the decoder. Here, we supply additional details. Let $(\cdot)_.$ denote tensor indexing. The mean representation that is fed to the decoder is found as

$$(\mathbf{z}_\mu)_{h,w} = (\boldsymbol{\mu}_{\mathcal{Q}|\mathcal{S}})_{hW+w}, \quad \mathbf{z}_\mu \in \mathbb{R}^{H\times W\times E} \ . \tag{10}$$

That is, the mean vector is *unflattened*. For the covariance, we encode the covariance between each point and its neighbours in an $N \times N$ window. First, the covariance output from the GP is unflattened,

$$\boldsymbol{\Sigma}_{\mathcal{Q}|\mathcal{S}} = \text{Unflatten}(\boldsymbol{\Sigma}_{\mathcal{Q}|\mathcal{S}}) \in \mathbb{R}^{H\times W\times H\times W} \ . \tag{11}$$

For each spatial location (k, l), we find the posterior covariance to the neighbouring location $(k + i, l + j)$,

$$(\mathbf{z}_\Sigma)_{k,l,iN+j} = (\tilde{\boldsymbol{\Sigma}}_{\mathcal{Q}|\mathcal{S}})_{k,l,k+i,l+j} \ , \tag{12}$$

$$\text{where} \quad (i,j) \in \{-(N-1)/2, \ldots, (N-1)/2\}^2 \ . \tag{13}$$

That is, the channels of $\mathbf{z}_\Sigma$ contains the covariance with respect to all neighboring locations in an $N \times N$ window.

F Qualitative Results

We provide qualitative results on PASCAL-5^i and COCO-20^i. First, we compare the our final approach to a baseline on the COCO-20^i benchmark. The baseline also relies on dense GPs – but uses a linear kernel, does not utilize the predictive covariance or learn the GP output space, and uses a single feature level (stride 32). Our final approach instead adopts the SE kernel, adds the predictive covariance in a local 5×5 region, learns the output space, and employs dense GPs at two feature levels (stride 16 and stride 32). The results are shown in Fig. 4. Our final approach significantly outperforms the baseline in these examples, making only minor mistakes.

In Fig. 5 we show qualitative results of our final approach on the PASCAL-5^i dataset. Our approach accurately segments the class of interested, even details such as sheep legs. In Fig. 6, we show results on COCO-20^i.

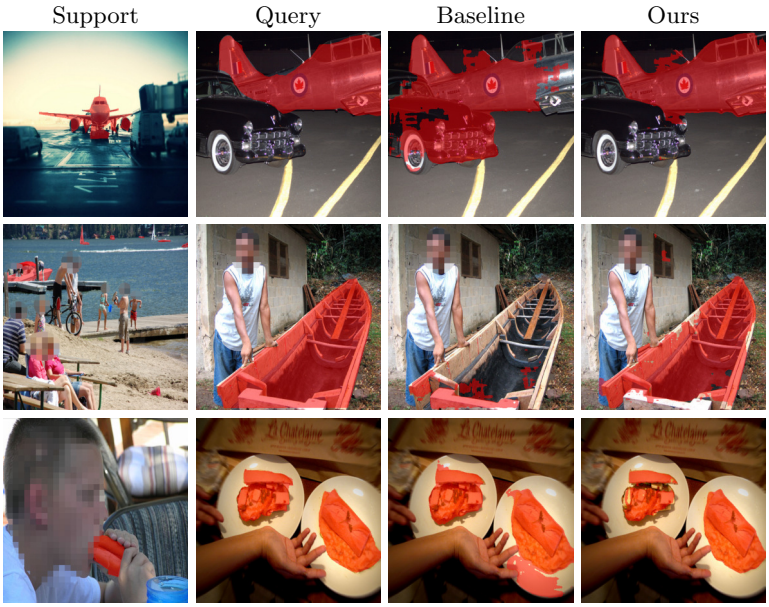

Fig. 4. Qualitative comparison between our final model and baseline in the 1-shot setting from the COCO-20^i benchmark. Human faces have been pixelized in the visualization, but the model makes predictions on the non-pixelized images.

Fig. 5. Qualitative results on challenging episodes in the 1-shot setting from the PASCAL-5^i benchmark.

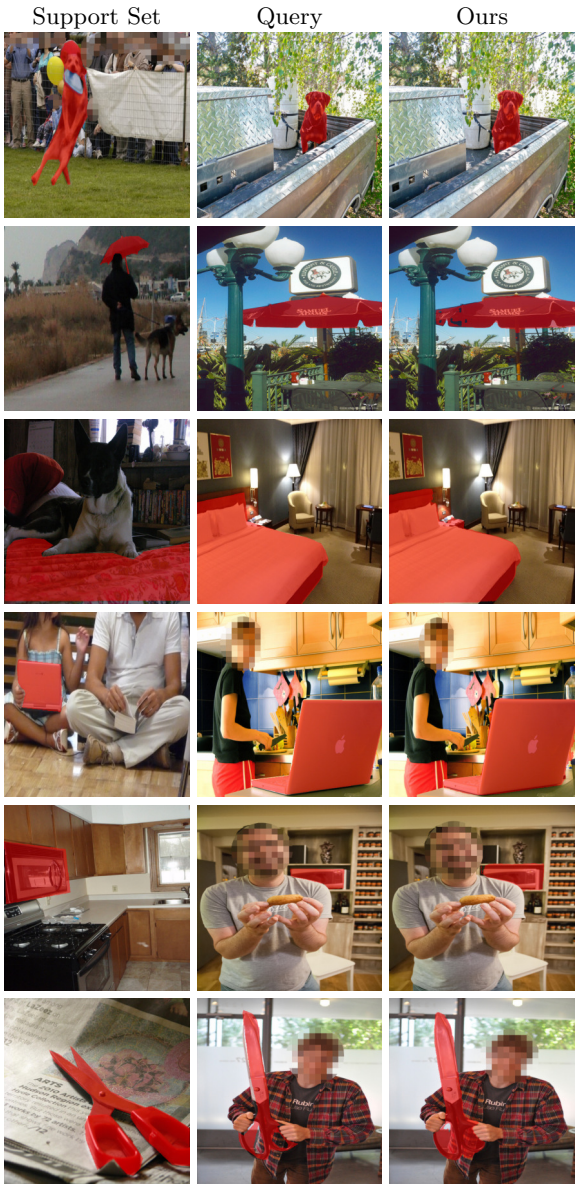

Fig. 6. Qualitative results on challenging episodes in the 1-shot setting from the COCO-20^i benchmark. Human faces have been pixelized in the visualization, but the model makes predictions on the non-pixelized images.

References

1. Allen, K., Shelhamer, E., Shin, H., Tenenbaum, J.: Infinite mixture prototypes for few-shot learning. In: International Conference on Machine Learning. pp. 232–241. PMLR (2019)
2. Azad, R., Fayjie, A.R., Kauffmann, C., Ben Ayed, I., Pedersoli, M., Dolz, J.: On the texture bias for few-shot CNN segmentation. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 2674–2683 (2021)
3. Bhat, G., Johnander, J., Danelljan, M., Khan, F.S., Felsberg, M.: Unveiling the power of deep tracking. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 483–498 (2018)
4. Boudiaf, M., Kervadec, H., Masud, Z.I., Piantanida, P., Ben Ayed, I., Dolz, J.: Few-shot segmentation without meta-learning: A good transductive inference is all you need? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13979–13988 (2021)
5. Calandra, R., Peters, J., Rasmussen, C.E., Deisenroth, M.P.: Manifold Gaussian Processes for regression. In: Proceedings of the International Joint Conference on Neural Networks (2016). https://doi.org/10.1109/IJCNN.2016.7727626
6. Dong, N., Xing, E.P.: Few-shot semantic segmentation with prototype learning. In: British Machine Vision Conference 2018, BMVC 2018 (2019)
7. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (VOC) challenge. International Journal of Computer Vision (2010). https://doi.org/10.1007/s11263-009-0275-4
8. Hariharan, B., Arbeláez, P., Bourdev, L., Maji, S., Malik, J.: Semantic contours from inverse detectors. In: Proceedings of the IEEE International Conference on Computer Vision (2011). https://doi.org/10.1109/ICCV.2011.6126343
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
10. Hu, T., Yang, P., Zhang, C., Yu, G., Mu, Y., Snoek, C.G.: Attention-based multi-context guiding for few-shot semantic segmentation. In: Proceedings of the AAAI conference on artificial intelligence. vol. 33, pp. 8441–8448 (2019)
11. Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings (2015)
12. Li, G., Jampani, V., Sevilla-Lara, L., Sun, D., Kim, J., Kim, J.: Adaptive prototype learning and allocation for few-shot segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8334–8343 (2021)
13. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2117–2125 (2017)
14. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: European Conference on Computer Vision. pp. 740–755. Springer (2014). https://doi.org/10.1007/978-3-319-10602-1_48
15. Liu, L., Cao, J., Liu, M., Guo, Y., Chen, Q., Tan, M.: Dynamic extension nets for few-shot semantic segmentation. In: Proceedings of the 28th ACM international conference on multimedia. pp. 1441–1449 (2020)

16. Liu, W., Zhang, C., Lin, G., Liu, F.: CRNet: Cross-Reference Networks for Few-Shot Segmentation. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 4164–4172 (2020). https://doi.org/10.1109/CVPR42600.2020.00422
17. Liu, Y., Zhang, X., Zhang, S., He, X.: Part-Aware Prototype Network for Few-Shot Semantic Segmentation. In: European Conference on Computer Vision. pp. 142—-158 (2020). https://doi.org/10.1007/978-3-030-58545-7_9
18. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
19. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=Bkg6RiCqY7
20. Lu, Z., He, S., Zhu, X., Zhang, L., Song, Y.Z., Xiang, T.: Simpler is better: Few-shot semantic segmentation with classifier weight transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 8741–8750 (2021)
21. Min, J., Kang, D., Cho, M.: Hypercorrelation squeeze for few-shot segmentation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6941–6952 (2021)
22. Nguyen, K., Todorovic, S.: Feature weighting and boosting for few-shot segmentation. In: Proceedings of the IEEE International Conference on Computer Vision. vol. 2019-Octob, pp. 622–631 (2019). https://doi.org/10.1109/ICCV.2019.00071
23. Patacchiola, M., Turner, J., Crowley, E.J., Storkey, A.: Bayesian meta-learning for the few-shot setting via deep kernels. In: Advances in Neural Information Processing Systems (2020)
24. Rakelly, K., Shelhamer, E., Darrell, T., Efros, A., Levine, S.: Conditional networks for few-shot semantic segmentation. In: 6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings (2018)
25. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning (2006). https://doi.org/10.7551/mitpress/3206.001.0001
26. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. IJCV pp. 1–42 (April 2015). https://doi.org/10.1007/s11263-015-0816-y
27. Salakhutdinov, R., Hinton, G.: Using deep belief nets to learn covariance kernels for Gaussian processes. In: Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference (2009)
28. Shaban, A., Bansal, S., Liu, Z., Essa, I., Boots, B.: One-shot learning for semantic segmentation. In: British Machine Vision Conference 2017, BMVC 2017 (2017). https://doi.org/10.5244/c.31.167
29. Siam, M., Oreshkin, B., Jagersand, M.: AMP: Adaptive masked proxies for few-shot segmentation. In: Proceedings of the IEEE International Conference on Computer Vision. vol. 2019-Octob, pp. 5248–5257 (2019). https://doi.org/10.1109/ICCV.2019.00535
30. Snell, J., Zemel, R.: Bayesian few-shot classification with one-vs-each pólya-gamma augmented gaussian processes. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=lgNx56yZh8a
31. Tian, Z., Zhao, H., Shu, M., Yang, Z., Li, R., Jia, J.: Prior Guided Feature Enrichment Network for few-shot segmentation. IEEE Transactions on Pattern Analysis & Machine Intelligence (2020). https://doi.org/10.1109/tpami.2020.3013717

32. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
33. Wang, H., Yang, Y., Cao, X., Zhen, X., Snoek, C., Shao, L.: Variational prototype inference for few-shot semantic segmentation. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 525–534 (2021)
34. Wang, H., Zhang, X., Hu, Y., Yang, Y., Cao, X., Zhen, X.: Few-Shot Semantic Segmentation with Democratic Attention Networks. In: European Conference on Computer Vision. pp. 730–746 (2020). https://doi.org/10.1007/978-3-030-58601-0_43
35. Wang, K., Liew, J.H., Zou, Y., Zhou, D., Feng, J.: PANet: Few-shot image semantic segmentation with prototype alignment. In: Proceedings of the IEEE International Conference on Computer Vision. vol. 2019-Octob, pp. 9196–9205 (2019). https://doi.org/10.1109/ICCV.2019.00929
36. Wilson, A.G., Hu, Z., Salakhutdinov, R., Xing, E.P.: Deep kernel learning. In: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016 (2016)
37. Wu, Z., Shi, X., Lin, G., Cai, J.: Learning meta-class memory for few-shot semantic segmentation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 517–526 (2021)
38. Xie, G.S., Liu, J., Xiong, H., Shao, L.: Scale-aware graph neural network for few-shot semantic segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5475–5484 (2021)
39. Xie, G.S., Xiong, H., Liu, J., Yao, Y., Shao, L.: Few-shot semantic segmentation with cyclic memory network. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7293–7302 (2021)
40. Yang, B., Liu, C., Li, B., Jiao, J., Ye, Q.: Prototype Mixture Models for Few-Shot Semantic Segmentation. In: European Conference on Computer Vision. pp. 763–778. Springer (2020)
41. Yang, L., Zhuo, W., Qi, L., Shi, Y., Gao, Y.: Mining latent classes for few-shot segmentation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 8721–8730 (2021)
42. Yang, Y., Meng, F., Li, H., Wu, Q., Xu, X., Chen, S.: A new local transformation module for few-shot segmentation. In: International Conference on Multimedia Modeling. pp. 76–87. Springer (2020)
43. Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N.: Learning a Discriminative Feature Network for Semantic Segmentation. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2018). https://doi.org/10.1109/CVPR.2018.00199
44. Zhang, B., Xiao, J., Qin, T.: Self-guided and cross-guided learning for few-shot segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8312–8321 (2021)
45. Zhang, C., Lin, G., Liu, F., Guo, J., Wu, Q., Yao, R.: Pyramid graph networks with connection attentions for region-based one-shot semantic segmentation. In: Proceedings of the IEEE International Conference on Computer Vision. vol. 2019-Octob, pp. 9586–9594 (2019). https://doi.org/10.1109/ICCV.2019.00968
46. Zhang, C., Lin, G., Liu, F., Yao, R., Shen, C.: CANET: Class-agnostic segmentation networks with iterative refinement and attentive few-shot learning. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2019). https://doi.org/10.1109/CVPR.2019.00536

47. Zhang, G., Kang, G., Yang, Y., Wei, Y.: Few-shot segmentation via cycle-consistent transformer. Advances in Neural Information Processing Systems **34** (2021)
48. Zhang, X., Wei, Y., Yang, Y., Huang, T.S.: Sg-one: Similarity guidance network for one-shot semantic segmentation. IEEE Transactions on Cybernetics **50**(9), 3855–3865 (2020)